Appendix G
Turtle Graphics

Turtle graphics were originally developed as part of the children's programming language LOGO by Wally Feurzeig, Daniel Bobrow, and Seymour Papert at MIT in the late 1960s [**AD86**]. Turtles went on to play a key role in the "constructionist learning" philosophy proposed by Papert in his seminal textbook *Mindstorms* [**Pap20**].

A turtle moves across the screen with a pen in its teeth (`https://www.turtleconservationsociety.org.my/do-turtles-have-teeth/`). Commands direct the turtle to lift the pen or lower it onto the screen, turn some number of degrees left or right, or move a specified distance forwards or backwards.

At any given moment, the turtle is located at a specific position, specified by (x, y) coordinates. The origin, $(0, 0)$, is at the center of the window, which is also the turtle's 'home'.

Another turtle attribute is its heading – the direction it is currently facing. The turtle's initial heading is 0 degrees, which is due east. The degrees of the heading increase as it turns to the left, so 90 degrees is due north.

In addition to its position and heading, other turtle settings include the pen's drawing color, the width of the line that is drawn, and whether the pen is up or down (i.e. is drawing or not).

The Python turtle documentation is excellent (`https://docs.python.org/3/library/turtle.html`), but we've included two tables of the commands that we've used the most (Tables G.1 and G.2) at the end of this appendix. They are by no means comprehensive, but cover the important functions.

G.1 Drawing Polygons

Almost all of our turtle code starts in the same way as Listing G.1 (drawPolys.py):

```
if __name__ == "__main__":
  rotAngle = int(input("angle? "))

  # create a turtle
  t = Turtle()
  t.hideturtle()
  t.speed(0)
  scr = t.getscreen()
  scr.title('Draw Patterns')

  polyCircles(t, rotAngle)

  scr.exitonclick()
```

Listing G.1. The start of a turtle program

An invisible, fast-moving turtle called *t* is created, and a reference is passed to the drawing function(s) (in this case, polyCircles()). A reference to the window is obtained as *scr*, and this is used to set the title and to keep the screen (the window) open after the drawing is finished. exitonclick() returns when it detects a mouse click inside the window, and lets the program finish.

polyCircles() calls rotatePolys() four times to draw a series of polygons rotated through 360 degrees.

```
from turtle import Turtle

LENGTH = 100

def drawPolygon(t, numSides):
  # draw a regular polygon with a specified no. of sides
  lineLen = LENGTH/(2*math.sin(math.pi/numSides))
  if (numSides > 2) and (lineLen > 0):
    angle = 360 / numSides
    for i in range(numSides):
      t.forward(lineLen)
      t.right(angle)

def rotatePolys(t, numSides, rotAngle):
  # draw a series of polygons in a circle,
  # spaced out at rotAngle degrees
  numShapes = 360 // rotAngle
  for shape in range(numShapes):
    drawPolygon(t, numSides)
    t.right(rotAngle)

def polyCircles(t, rotAngle):
  # draw four circles of polys in different colors
  t.pensize(3)
  t.pencolor('blue')
  rotatePolys(t, 3, rotAngle) # triangles
  t.pencolor('red')
  rotatePolys(t, 4, rotAngle) # squares
  t.pencolor('green')
  rotatePolys(t, 5, rotAngle) # penatgons
  t.pencolor('orange')
  rotatePolys(t, 6, rotAngle) # hexagons
```

Listing G.2. Drawing polygons

The end result is shown in Fig. G.1. This was produced after the user entered 10 degrees as the value for `rotAngle`.

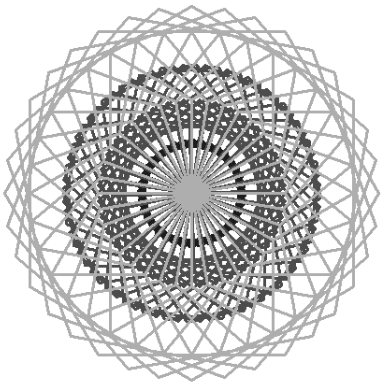

Figure G.1. Circles of polygons

G.2 Responding to Mouse Clicks

Another common top-level form for a turtle program is to use onclick() and mainloop() to listen for mouse clicks to trigger function calls (Listing G.3; drawClick.py).

```
from turtle import Turtle

def drawClick(x, y):
  t.goto(x, y)
  t.write(str(x)+","+str(y))

t = Turtle()
t.hideturtle()
t.speed(0)
scr = t.getscreen()
```

```
scr.title('DrawToClicks')
scr.onclick(drawClick)
scr.mainloop()
```

Listing G.3. Responding to mouse clicks

A mouse click causes drawClick() to execute, which moves the turtle from its current position to the click point, drawing a line in the process, and writing the coordinate details to the window (see Fig. G.2). Note, that it's not possible to pass a turtle reference to drawClick(), so *t* must be global.

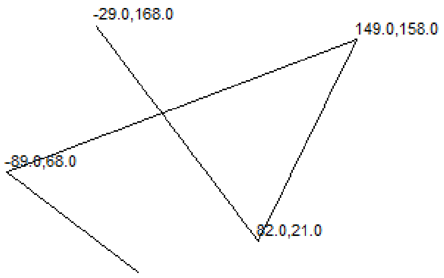

Figure G.2. Lines drawn by clicking

This approach means that the program must be terminated by clicking on the turtle window's close box.

G.3 Beautiful Recursion

Listing G.4 (spirals.py) shows that recursion is easy to illustrate with turtle graphics. The patterns in Fig. G.3 were generated by calling spiral() with slightly different arguments (some examples are commented out in the code).

```
def spiral(t, sideLen, angle, scaleFactor, minLength):
  # Draw a spiral recursively."""
  if sideLen >= minLength:
    t.forward(sideLen)
    t.left(angle)
    spiral(t, sideLen*scaleFactor, angle,
                scaleFactor, minLength)

t = Turtle()
t.hideturtle()
t.speed(0)
```

```
scr = t.getscreen()
scr.title('Draw Spiral')
# spiral(t, 200, 90, 0.8, 10)
# spiral(t, 200, 72, 0.97, 10)
# spiral(t, 200, 80, 0.95, 10)
# spiral(t, 200, 121, 0.95, 15)
spiral(t, 200, 95, 0.93, 10)
scr.exitonclick()
```

Listing G.4. Draw a spiral

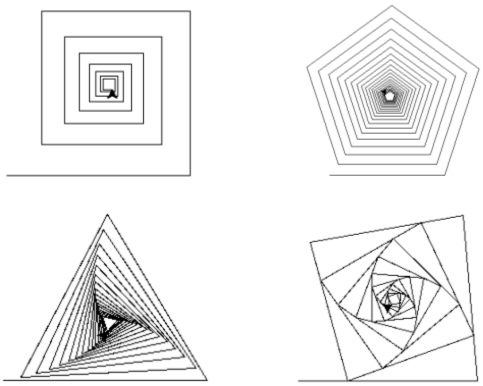

Figure G.3. Some possible spirals

G.4 Multiple Turtles

Many turtle examples don't bother creating an explicit turtle, *t*, as we have done, because a default turtle is always available. We prefer the explicit approach since it makes it clearer what's being manipulated by our functions.

It's possible to create multiple turtles, as in Listing G.5 (drunks.py), which also shows Python's random module making turtle behavior less predictable.

```
MAX_STEPS = 50
STEP_SIZE = 50
colors=["red","orange","green","blue","purple","violet"]

n = int(input("n? "))
# create n turtles
ts = [ Turtle() for _ in range(n)]
for i in range(n):
  ts[i].setheading(360*random.random())
  ts[i].speed(0)
  ts[i].pencolor(colors[i % len(colors)])
scr = ts[0].getscreen()
scr.title('Drunk Turtles')

# move randomly for max steps
for t in range(MAX_STEPS):
  for i in range(n):
    ts[i].left(360 * random.random())
    ts[i].forward(random.randint(STEP_SIZE//5, STEP_SIZE))

scr.exitonclick()
```

Listing G.5. Drunk turtles on the move

When the user creates five turtles (each with a different colored pen), something like Fig. G.4 is generated.

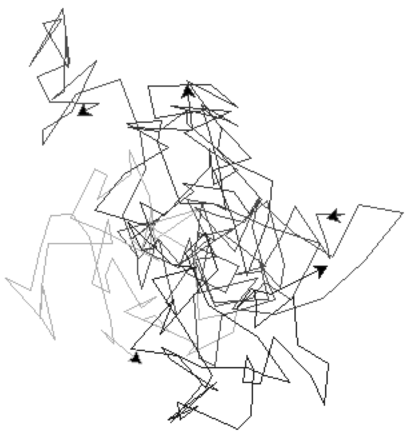

Figure G.4. Five drunk turtles

G.5 LOGO Flowers

The last program of this section produces the most complex turtle drawing we've seen (Fig. G.5), although it only uses basic features.

The program was originally written in LOGO by William Weinreb for *BYTE Magazine*, Nov. 1982 (`https://archive.org/details/byte-magazine-1982-11/page/n119/mode/2up`). We've translated the code into Python, which you can find in flowers.py.

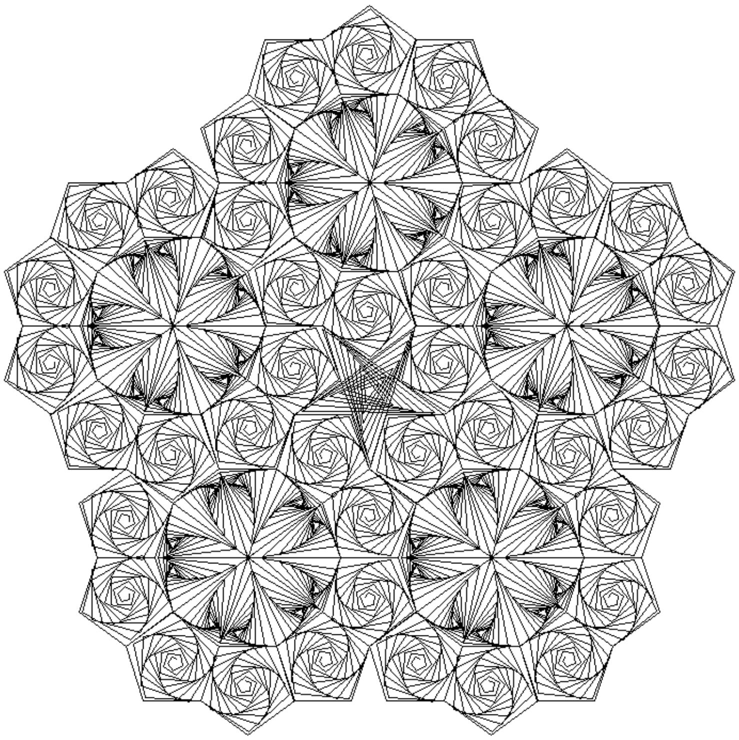

Figure G.5. A bunch of flowers

Turtle Methods	Description
t = Turtle()	Creates a new Turtle object and opens its window.
t.penup(); t.pendown()	Raises/lowers *t*'s pen from/to the drawing surface.
t.isdown()	Returns True if *t*'s pen is down, otherwise False.
t.left(degrees); t.right(degrees)	Rotates *t* to the left or right by the specified degrees.
t.forward(distance)	Moves *t* the specified distance in the current direction.
t.backward(distance)	Moves *t* the specified distance in the opposite of the current direction.
t.hideturtle(); t.showturtle()	Makes *t* invisible or visible (but doesn't affect the pen).
t.goto(x, y)	Moves *t* to the specified position.
t.position()	Returns the current (x, y) position of *t*.
t.xcor(); t.ycor()	Returns the turtle's x or y coordinate.
t.heading()	Returns the current direction of *t*.
t.setheading(degrees)	Points *t* in the indicated direction, which is specified in degrees. East is 0 degrees, north is 90 degrees, west is 180 degrees, and south is 270 degrees.
t.home()	Moves *t* to the center of the window and then points *t* east.
t.pencolor(r,g,b); t.pencolor(string)	Changes the pen color of *t* to the specified RGB value or string, such as 'red'. Returns the current color of *t* when the arguments are omitted.
t.pensize(width)	Sets the pen's drawing line to the given width.
t.speed(s)	Sets the turtle's speed: 1 is slowest, 10 is fast, 0 is the fastest.
t.write(message)	Writes message at *t*'s current position.

Table G.1. Commonly used turtle functions

Screen Methods	Description
scr = t.getscreen()	Returns a reference to the turtle's drawing window.
scr.setup(width, height, x, y)	Sets the size and location of the window.
scr.setworldcoordinates(x1, y1, x2, y2)	Sets the coordinates of the window to $(x1, y1)$ at the lower left and $(x2, y2)$ at the upper right.
scr.exitonclick()	Causes the window to close when clicked.
scr.onclick(function)	Sets up an automatic call to a function whenever a mouse click is detected; the coordinates of the click are passed to the function.
scr.mainloop()	Makes the program enter an event processing loop to listen for mouse clicks and other events.

Table G.2. Commonly used turtle screen functions